CSCI 36200: Data StructuresProgramming Assignment 2Instructor: Dr. Snehasis Mukhopadhyay

Due date: April 19, 2018

# Project Description

The program compares different sorting algorithms together. This program takes the data from a file called n.txt, in this file all the numbers has been coma separated and then the program parses these numbers using the coma and compares each number with the help of different algorithms like “quick sort”, “heap sort”, “merge sort”, “insertion sort”.

Objective 1:

The project uses a file with 100 randomly generated numbers exist from 1 to 20000 numbers and when those numbers were obtained they were put in a file called “n.txt”. Some of these numbers can repeat since they are completely random.

Objective 2:

The algorithms which I will be using are “quick sort”, “heap sort”, “merge sort”, and “insertion sort”. They are in their own separate files with their header files and the code is completely reusable if ever want to create another file for a different sorting algorithm.

Objective 3:

Each element will be sorted that are placed in the file n.txt. The program only works numbers. In this example the numbers are from a randomly created list.

Objective 4:

Execution time

Heap sort: 2*N* log *N* − *O*(*N* log log *N*).

Merge sort: *N* log *N* + *N*

Insertion sort:

Assume that the length of array is N

Assume that the comparision and swap is O(1)

The outer loop run N - 1 times

i = 1: the inner while run maximum 1 times

i = 1: the inner while run maximum 2 times

i = N - 1: the inner while run maximum N - 1 times

The total is 1 + 2 + ... + N - 1 = N \* (N - 1) / 2

So the complexity is O(N^2)

Quick sort:

Assume that the length of array is N

Assume that the comparision and swap is O(1)

The partition function for array with length = L takes O(L)

because it uses one for loop that run from start + 1 to end

The quickSort is recursive function.

If array has length is N, takes O(N) + 2 \* the complexity of quickSort on N / 2

(because it calls recursive function on the left part and on the right part with length = N/2)

It is O(N) + O(N) + ...

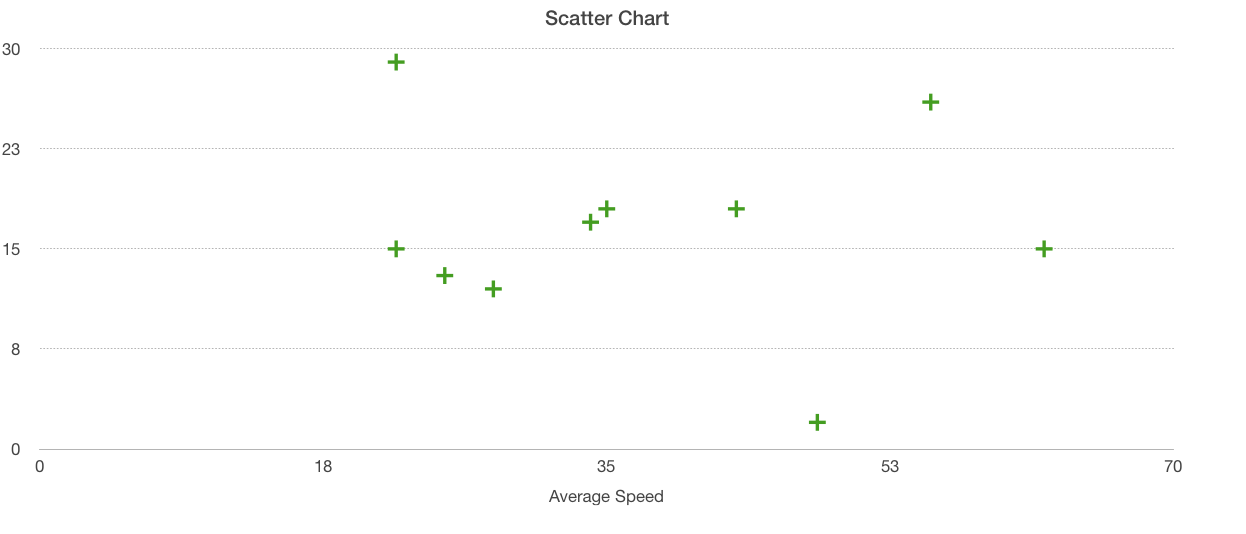
The level of recursive function is logN because for each call, the length is reduced a half.

So the total complexity is O(NlogN)

How the Algorithm works:

The algorithms work recursively. I found this method to be more efficient then an iterative method. The algorithms use the data from the file and compare and contrast with the elements in file and then switch and swap accordingly.

Before sorting:



After Sorting:

